Library of “JurInfoR”

Series: Fundamental Basics of Information Technologies

V. E. Wolfengagen

Applicative computing. Its quarks, atoms and molecules. Edited by Dr. L.Yu. Ismailova. — Moscow: “Center JurInfoR”, 2010. — 64 p.

This work covers the advanced topics in main ideas of computing in general. This material is approved in practice of NRNU MEPhI, MIPT and several other educational centers of the Russian Federation.

Its 1st part represents an outlook of computations, which is achieved by adoption of the atomic doctrine for specified reference system of primary objects. The main attention is given to finding-out of technological features of computations with objects. Their interaction is considered in applicative environment that allows finding out internal structure of usual operations which knowledge allows understanding their properties. The choice of initial constant entities, considered as primary and referred as combinators is discussed. These initial entities are used as the basic ``building blocks'', entering in applicative environment in interaction with each other. This interaction results in the constructs, giving representative sets of usual operators and to the embedded computing systems.

The 2nd part gives some supply of environments for educational and methodical complex of corresponding discipline (EMCD).

This material is suitable both for advanced learners and beginners in Computing and Information Technologies as well as in Discrete Mathematics (DM) and Fundamental Basics of Information Technologies (FBIT). It helps for developing the intuition sufficient for successful navigation across the dramatically changing world of innovative information processes which occurs both in nature and technology.

Material is especially useful for the instructor, postgraduate and graduate students of IT-specialties and is suitable for the system of training and advancing the qualification of specialists.

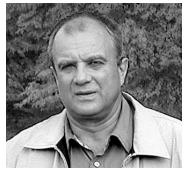
ISBN 978-5-89158-177-7

http://www.jurinfor.ru

V. E. Wolfengagen

Applicative computing
Its quarks, atoms and molecules

ЮрИнфоР®



V.E. Wolfengagen

Doctor of Technical Science, professor,
ACM Senior Member

ViacheslavWolfengagen received his Candidate of Technical Science degree in 1977 and the Doctor of Technical Science degree in 1990 from Moscow Engineering Physics Institute. He is a full professor of theoretical computer science and discrete mathematics at the Cybernetics Department of NRNU MEPhI and at the Faculty of Innovations and High Technologies of MIPT. Since 1994 he has been with the Institute for Contemporary Education "JurInfoR-MGU" in Moscow where he is currently a head of the Department of Advanced Computer Studies and Information Technologies.

He chaired the 1999-2009 International Workshops in Computer Science and Information Technologies (CSIT). He is author of the books *Logic: Techniques of Reasoning* (2001, Center "JurInfoR"), *Constructions in Programming Languages: Methods of Description* (2001, Center "JurInfoR"), *Categorical Abstract Machine: Introduction to Computations* (2002, Center "JurInfoR"), and *Combinatory Logic in Programming: Computations with Objects through Examples and Exercises* (2003, MEPhI - Center "JurInfoR").

His research interests include data models, database design, software development databases, object and object-oriented programming and design, computation theory, programming languages, applicative computational systems. He was a manager of research and development projects *Logical Applicative Modeling Base of DAta* LAMBDA (version 3, project 93-01-00943 granted by RFBR), *Categorical Object-Oriented Abstract Machine* COOAM (project 96-01-01923 granted by RFBR), *Metadata Objects for Proxy-Based Computational Environment* (project 99-01-01229 granted by RFBR).

Library of "JurInfoR"
Founded in 1994

V. E. Wolfengagen

Applicative computing

Its quarks, atoms and molecules

Moscow
"Center JurInfoR" Ltd.
2010

LBC 32.97
UDC 004
B721

Library of "JurInfoR"
Founded in 1994
Series: Fundamental Basics of Information Technologies

V. E. Wolfengagen

Applicative computing. Its quarks, atoms and molecules. Edited by Dr. L. Yu. Ismailova. — Moscow: "Center JurInfoR", 2010. — 62 p.

This work covers the advanced topics in main ideas of computing in general. This material is approved in practice of NRNU MEPhI, MIPT and several other educational centers of the Russian Federation.

Its 1st part represents an outlook of computations, which is achieved by adoption of the atomic doctrine for specified reference system of primary objects. The main attention is given to finding-out of technological features of computations with objects. Their interaction is considered in applicative environment that allows finding out internal structure of usual operations which knowledge allows understanding their properties. The choice of initial constant entities, considered as primary and referred as combinators is discussed. These initial entities are used as the basic "building blocks", entering in applicative environment in interaction with each other. This interaction results in the constructs, giving representative sets of usual operators and to the embedded computing systems.

The 2nd part gives some supply of environments for educational and methodical complex of corresponding discipline (EMCD).

This material is suitable both for advanced learners and beginners in Computing and Information Technologies as well as in Discrete Mathematics (DM) and Fundamental Basics of Information Technologies (FBIT). It helps for developing the intuition sufficient for successful navigation across the dramatically changing world of innovative information processes which occurs both in nature and technology.

Material is especially useful for the instructor, postgraduate and graduate students of IT-specialties and is suitable for the system of training and advancing the qualification of specialists.

ISBN 978-5-89158-177-7

NRNU MEPhI • "Center JurInfoR" Ltd. • MIPT

Part 1

Quarks, atoms, molecules of computing

Abstract. Computing and its development sets up a range of questions on the most part of which answers either are incomplete, or unknown. Some of them: what is a ‘computation’? What is an ‘information’? What is possible to learn, using computing? What cannot be learned, using computing? – have fundamental value.

In the present work the main attention is paid to finding-out of technological features of computations with objects. Their interaction is considered in applicative environment that allows to find out internal structure of usual operations knowing which allows to understand their properties. The choice of initial constant entities, considered as primary and referred as *combinators* is discussed. These initial entities are used as the basic “building blocks”, entering in applicative environment in interaction with each other. This interaction results in the constructs, giving representative sets of usual operators and of the embedded computing systems.

About the author. Prof. Wolfengagen V. E. (vew@jmsuice.msk.ru), the head of ACS & IT dept. at “JurInfoR”. He is working in an area of computing science and information technologies, including applicative computational systems, λ-calculus, combinatory logic, type systems. RFBR’s projects 93-01-00943-a (LAMBDA), 96-01-01923-a (COOAM), 05-01-00736-a.

Introduction

Computing and its development puts a lot of questions, on the most part of which answers either are incomplete, or unknown. Some of them: what is a ‘computation’? what is an ‘information’? what is possible to learn, using computing? what cannot be learned, using computing? – have fundamental significance.

These questions accompanied computing, since 1940th. It seemed, there are answers on them, but today the same

questions are also indicated by all and everywhere, in all areas of a science, engineering, business and even policy.

Long time there was a tradition according to which computing was considered as a science about the phenomena accompanying computers, and this sight did not raise the doubts. Computing always was and remains a science about information processes. Starting approximately with 1995, experts from different areas of a science, one behind another, began to declare, that they, in their area, find out natural information processes. These openings have introduced other tradition according to which computing began to be considered as a science about both natural and artificial simultaneously into use.

By old tradition computing was most naturally described by ideas from *basic technologies* – programming, graphics, networks and supercomputing. The present tradition urgently demands to express computing in terms of fundamental principles or even to deduce computing from some fundamental principles. If one deduced computing from principles not only its deep structures will be opened, but also their applicability in other areas of a science will be cleared up as well. Thus the general aspects of distinct technologies also are opened, creating opportunities for innovations. At the same time essentially new ways of stimulation in many respects lost interest to computing among youth are opened.

In 1940th the computations were considered simply as a tool for decision of the equations, decodings of codes, the analysis of data and management of business processes. But in 1980th computing have developed up to such degree that have turned to a new scientific method, connecting traditional understanding of a theory with experiment. And in 1990th a shift in understanding of a role and a place of computing followed, as many researchers in different scientific areas have

come to conclusion, that they have collided with information processes in natural science deep structures. For example, these are quantum effects in physics, DNA in biology, thought processes in cognitology, streams of information in economy. Computations were included into lives together with new ways of a decision of the problems, new forms of art, music, cinema, and also together with new forms of the commerce, new approaches to training and even new slang on which began to speak.

The fundamental questions designated right at the beginning of applying computing became important and in a variety of those areas in which people in the work essentially lean on computation and computing methods. Actually, studying of aspects of computing, bringing the greatest advantage in traditional sciences, most of all helps an establishment of fundamental bases and principles of the computing.

Metaphorical speech turns of daily speech have replenished with phrases like: “I am programmed on such behavior” or “my brains failed and need to reboot”. Active *germination* of computing in all the spheres of a daily life has led even to that from students of the Washington university began to demand “fluent possession of information technologies”, that assumes their good knowledge and skill to apply in various situations. There was a need and requirements to “computational thinking”. This assumes usage of principles of computing both in a science, and in a life. Computations have got everywhere, and also began to be found out everywhere.

At an establishment of principles and explanations of the event, proceeding from them, there is one more advantage in comparison with technologies: it is easier to learn the principles, than the technologies. The description of a field of activity in the language of technologies well worked earlier when the amount of known base technologies was not so big. For

example, the Association of Computing Machinery in 1989 totaled 9 base technologies, and in 2001 their amount became already 36, forming 630 direct interrelations that became problematic for direct studying on former educational patterns.

For today while it is necessary to ascertain, that computing have not became to be expressed in terms of fundamental principles, it still remains prescriptional and technological. In other sciences there was other situation, in them the circle of the phenomena is established, proceeding from the established base principles. It testifies to a known maturity of such sciences, but not computing which only just reaches a status in its growth, being in which it is possible to speak about its principles.

1 Invariants of computing

The establishment of principles of computing represents an uneasy process at all. When they in computing were accepted to business and it began to be developed intensively then its essence seemed by itself understood, but its forms were produced. The boom which has arisen in 1970th of the *models of computations* does not stop on present time. In the beginning of this boom nothing constrained the developers, and opening opportunities promised boundless prospects.

Nevertheless from time to time arose and there is a question as computing is arranged, but from it more often simply waved away. Similar interest looks purely academic, and according to occurring opinion, information and computing technologies are a destiny of greater companies and powerful collectives. But the general picture of computing only will win, if it will be possible in it to find out those *invariants*, which are kept from change of forms.

Invariants play a role of global constants, and from the computing point of view – primitive 'building blocks', using

with which it is possible to design this or that ‘world’ of computing. As on each of available forms of computing its developers precisely and rigidly declare the rights, both developers and users zealously defend the world designed by them. They concern to this world, as by environment of the information dwelling, stopping attempts of intrusion into it from the outside and if it was possible to establish unity of forms it would promote improvement of mutual understanding in the diversity of information community.

It was known that similar invariants exist. The question is put differently, as how to use them to take advantage. Really, it is necessary to recollect about *combinators*, rather recently opened in the world of metamathematics, as it supports confidence of an available unity of forms. From the intuitive point of view an *environment* of computations contains all or nearly so everything, that concerns to construction of a *result*: there are variables and their actual values in it, and they are carried not only positionally, but also contextually.

Everything that is done in computing, can be dropped down to some fundamental principle which many admit, and which is not rejected by overwhelming majority: consider the *identifier* and *relative to environment* associate for it some *construct* which will be considered as its *value*. Computation considers this process of constructing, and computing develops technologies of implementation of the construction. So, the correspondence between the identifier and its value is parameterized by environment. The number of all identifier-value possible combinations is so great and impressive, that it is not feasible to construct the hypothetical table, using which all the possible instances concerning the identifiers and their values are listed. Certainly, for realization of similar strategy of computation one should get a huge database which would be in a status of permanent upgrade and updating. At a

modern level of understanding it is considered technologically unacceptable. From the positions of a theory of databases and relational model this relation is considered as *explicit* transfer of all possible combinations of individuals in which they can appear, remaining within the limits of this relation. From a mathematical point of view the point under discussion is a mapping for which are *in advance known* the range of definition – its *domain*, – and the range of value – its *range*. They are not only great in volume, but also are subject to changes, and development of a theory of mappings with variable domains-ranges is in embryo status. D.Scott in (D. S. Scott, [17]) has suggested to consider constructions of *variable domains*, but in the field of computer sciences it was not widely supported, as burst a little bit later the boom of computing prompted other promising prospects at once in many directions, and for reception of fruits it was not necessary to bring a burdensome payment of development the complex and interdisciplinary theory, which efficiency should be defended in addition.

In combinatory logic argued differently, believing, that it is necessary to borrow in designing actual mappings, not caring about existence of their ranges of definition and ranges of value and due to efforts of M. Schönfinkel (M. I. Schönfinkel, [15]) and H. Curry (H. B. Curry, [6]) such a theory has been developed. More radical intention consisted of finding the minimal set of mathematical entities, using with which it would be possible to design all building of modern mathematical knowledge as other forms of knowledge were not considered but believed having no right on that with them seriously were considered. It is important, that opened by them *combinators* underlay any mathematical and metamathematical reasoning. Later, in a process of developing the information technologies, their fundamental value for computing has been known.

In traditional computing a central concept, without which as usually it is considered, it is impossible to operate, is the representation of a *variable*. The variable plays a role of 'numbers in general', that helps to build the general statements and to analyze their properties. At once it has been realized, that variables should be considered more widely, including the 'entities in general', some indeterminants, not reducing their sphere of action only to pure arithmetics. Combinators are perceived as 'constants in general', assuming, that the structure of knowledge, by its organizing, is granulated by constants. At the same time it is not unexpected, that they either in mathematics, or in computing have no reliable definitions neither of a variable, nor a constant. Such situation undermines trust to the fundamental trues saved up in these areas which formal expression with necessity is grounded both on constants and on variables.

A special attitude to variables as to constructs which in any way are impossible to be avoided, has generated the modern reality of information technologies expressed in programming languages. Predilection for variables generates difficulties of basic nature when the question of application of computing arises. Fitting the knowledge within a form is carried out in *books*, for which the way of expression and organizing is considered so well-known, that, actually, is not exposed to studying. An exception is the project Automath by de Bruijn (N. G. de Bruijn, [8]). Having started in 1967, it pursued the aim to develop the environment for expressing the mathematical theories in a form suitable for computer check of their correctness. A hypothesis, that if a statement is expressed correctly then it is correct in fact, was laid down in its basis. Any other norms of correctness were not introduced. There is no need to forget, that 'correct' means simply 'based on rules' and it remains to understand just a question, what a *rule* is,

but this is the most debatable question not only in modern metamathematics, but also in computing.

It is considered, that a language by its structure forms *logic*, that immediately leads to necessity to clarify its fundamental difficulties which though are known, but are not considered quite perceived. Presumably neither logics, nor the mathematical grounds were not used in Automath. In this project all the mathematical material formed the *books*, written in a language, and the language was based on lambda-calculus with types, in terms – in the *rules*, – of which representations about 'definition', 'theorem', 'proof', 'axiom' were expressed. The book represents a construction consisting of nested blocks, and opening of the block corresponds to introduction of a *typed variable*. Variables present mathematical objects or mathematical proofs, and the system of their mutual correspondences is developing quite similarly. In other words, an idea of *proof-as-object* was incorporated in this project from the very beginning.

The only interpretation of this block structure is on a share of logic metameans. Is it a lot of or not – sets up an open question. Is it enough to introduce in the expressed in such a way text the fundamental difficulties burdening metamathematics? However, by a form of its expression the project represents an innovative way to consider relation of logic with mathematics, at least, from positions of possibilities of modern computing. Processing the books which are expressing the mathematical knowledge in a language of Automath, becomes attractively easy and natural for dominating general mathematical practice. It gives some didactic and educational opportunities: the teaching of mathematics so that students could study it, receives a sound ground in the available and developing information technologies. Moreover, teaching from a category of art passes into a category of technology

when it is enough “to explain” the machine – in a language of the project, - the constructive organization of the text by its form, but not by its meaning. Thus any preliminary logic or mathematical “implied sense” is not brought in, and the validation of correctness of the text is assigned to the environment of computing. On a plan, this approach does not cover the automation neither of a mathematical invention process, nor of search of the proof process for the theorems having been formulated: Automath plays a role of the attentive reader of the material which is correctly represented by its form.

As it has appeared, it was required to develop the *virtual reader* in computing which will correctly carry out a process of reading the *virtual book* which is correctly applied by the appropriate form. It is caused by extreme complication of mathematicalized knowledge when its mastering or estimation of correctness exceed usual human abilities. But not possibilities, as, taking to the aid computing and its environment, knowledge turns in from *actual* to *possible* one. In the newest information technologies it has renewed heightened interest to semantic networks (T. Berners-Lee et al., [3]).

2 New paradigms of computing

The sense of a term ‘computing’, maintained nowadays though with changes, but corresponds to that understanding which has been accepted 60 years ago at establishing of the ACM (Association for Computing Machinery). The shifts which have outlined at the newest time in its understanding appear rather essential, having three general characteristics: computing is not necessarily carried out only on the basis of technology of silicon integrated schemes; the basic computing elements are implemented physically, becoming not simply a theory; transition to the new thresholds of miniaturization often making

from 1 up to 100 nanometers is carried out. Under these conditions the customary representations about computing start to be reconsidered, but this does not mean at all refusal of available representations or technologies.

Some of new forms of realization of computing have the expressed addressing and are intended for the decision of quite certain problems, for example, having the raised computing complexity or concerning particular applications. Though practical and daily application of a majority of them is only ahead, expecting occurrence of suitable devices, their modeling originality carries away its natural fundamentality, innovation and potential. The opportunity to create a basis for new forms of processing of the information opens in the latter case and, being based on them to develop families of applications. Discussion of their technological opportunities occurs usually outside of sphere of the periodic literature on computer sciences, and research is moved to area of such natural sciences, as physics or chemistry. It is caused by a status of works which for the present time are at a level of study of basic ideas – or in the most initial technological phases, or at a level of experiments. In a process of technological ripening similar forms of computing get in a sight of computer sciences, starting to be used in practice. Realization of new forms of computing encounters difficulties of development of hardware, demanding development of new architectures. On the other hand, difficulties are caused also by attempts to equip new forms of computing by the suitable software as it is required to develop new schemes of the organization of computations and new algorithms. The situation is similar to that which was at transition from consecutive algorithms to algorithms for parallel computing systems. Some known decisions can be transferred on a new area, and others – cannot.

The known new forms can be subdivided into two greater groups. In the first of them are put the decisions based on *nanotechnologies*. They are the organization of schemes of computations on nanofibres, coal nanotubes, organic molecules, bio-DNA and quantum effects. In the second are put the special forms of computing including optical, micro/nanoliquid and chaotic computations.

3 Revision of computing foundations

The consumers in a customary computing have promoted in understanding of sets and have learned to maintain the models of computations based on the notion of a variable for which it is known, what domain it will range – *typed* models of computations, or models of computations with types. In other words, an idea of *type* has received a wide circulation and a universal recognition, and all available programming systems have to more or less extent worked out management systems of types of variables/objects.

The models based on classes remain less worked out as they conduct to construction of domains which elements are other domains in turn etc., and for such structures the volume of computations needed to evaluate true or false of statements sharply increases.

The models of computations, in which indication of type of variables was not supposed – the *untyped* models of computations, remain completely neither worked out nor comprehended in practice. The leading position among them is borrowed with λ-calculus and combinatory logic. Though λ-calculus also is recognized in practice of programming technologies, but not so fast, as it deserves that, the combinatory logic is applied obviously insufficiently. Combinators – the core primary elements of combinatory logic, – were introduced in

hope to get rid of arithmetic style of working with numerical data, characteristic for overwhelming majority both of former and existing programming systems, and instead of it to pass to other style of reasonings in terms of objects and their applications to each other. In use of the first style the *calculation* is carried out – from a words 'to compute numerical value', – and in use of the second one – *computing* in the true and self sense of this term. Besides that combinators deal with *free* variables which are understood as indeterminants and have no deal with *bound* variables at all. As a matter of fact, computing with combinators is carried out in terms of constants. It is even better to say: 'constant objects', and they are constants not in absolute sense, but in a *relative* sense when objects reveal the *property of a constancy* relatively the *environments* in which computing is carried out. It remains to formulate suitable definition of a constant – to give the characteristics to constancy property, – and also to be determined with a model of environment. This just appears uneasy business as influences all the elements of computing architecture without any exception.

Now we approach to an important point – necessity to formulate the *representation of a constant*, which would appear fruitful for computing in general. Certainly, it would be desirable to leave this representation both intuitively transparent and coordinated with an available natural-science representation of a constant.

4 Notion of a constant

Do we know a lot of or a little concerning what a "constant" is? To the essence, everything connected with representation about a constant has appeared subcontracted to the mathematics. And both representation about a constant, and representation about a variable are considered as self evident in it. At the best it

admits, that the constant is – unlike a variable, that does not vary, and the variable, in particular, can be a constant. These representations remained firm or seem those till a time. So would be now, if not computing.

It has appeared, that the best that was offered is a general agreement on a constant which other side was a variable.

At a discussion, discourse is usually conducted about values behind which numbers are there and then seen. Numbers and their processing – calculations, – worked for our advantage for a long time perfectly and trouble-free. Even in computers while their productivity has not grown so, that the technology has approached to the physical threshold of miniaturization allowing precisely fix and distinguish zeroes (0) and ones (1). And now there comes some critical point in understanding, what the computing is.

Whether is there a theory of constants? The answer to this question is known, even more: it appears, that there is *no* language on which it would be possible to speak about constants mathematically precisely. Attempts to speak about them in absolute sense look restricted and insufficiently grounded. In a *relative* sense it is possible to achieve greater: why not to name a constant the 'object' which does not depend on certain 'context'? Or, even better, it does not depend on 'environment' which is considered as some context for a while. We shall look, whether is there an adequate language for exact expression of this idea.

For a test we shall take λ-expression

$$(\lambda x.\text{object})(\text{environment}) = \text{object},$$

which, obviously, expresses the thesis of constancy: $(\lambda x.\text{object})$ can be considered as a unary constant function. But transition to λ-language has demanded introduction in a consideration the

representation about a ‘variable’ which also appears bound by the abstraction! In this moment all power of usual mathematics is received, but simultaneously all the known costs of operating by variables is inherited.

Let’s take expression of combinatory logic

$$K(\text{object})(\text{environment}) = \text{object},$$

to which should operate under the scheme of axioms K : $Kab = a$. To say in a language of combinatory logic, that the object is a constant one, that represents a constant relatively to environment, means, to apply the combinator K to it. Then – relatively to the given environment, and this is, possibly, the other object, – the ‘object’ is simply quoted, that now quite us arranges.

How precisely is to tell in mathematics, that the object varies? For this purpose we shall write down

$$V(\text{object})(\text{environment}) = \text{object}',$$

where V is some combinator which acts on the object interesting to us – old object, – relatively the given context and results in a new object – object$'$.

And again, it is necessary to bring in a traditional mathematics to record this idea in the λ-language. We write

$$V(\lambda x.\text{object})(\text{environment})(\text{object-1}) =$$

$$V(\text{object})(\text{environment})_{x:=\text{object-1}},$$

that is, the old environment has appeared to be replaced by a new one $(\text{environment})_{x:=\text{object-1}}$, differing from the old one unless that, instead of a variable x, substitution of the object ‘object-1’ is executed. There was generated an expression with

a variable which has to be served by a newly introduced rule of substitution with all the known consequences.

Thus, applicative on its plan λ-language appears burdened by all known technical difficulties of processing both free and bound variables. In a language of combinatory logic such a burden is not still present.

5 Functions

The usual idea, concerning functions, consists of that they are a special case of a law of correspondence putting in predetermined relation to the elements of one domain the elements from other domain. So first of all it is necessary to be determined with a domain A which is considered as represented by a constant object, that is its structure obviously does not include free variables. For domain A and combinator $\mathbf{B}$ we shall demand

$$A = A \circ A \equiv \mathbf{B}AA \equiv 1_A.$$

For mapping $f : A \to B$ where f is an object which does not contain free variables, we shall demand

$$f = B \circ f \circ A \equiv B \circ (\mathbf{B}fA) \equiv \mathbf{B}B(\mathbf{B}fA).$$

Thus, mappings f are determined as the triples (A, f, B).

6 Interaction of an object and environment

6.1 Environment

A thesis, that interaction of objects needs the intermediary – *environment*, is perceived as obvious one. At least, currently it does not attract doubts. More rigorously, to initialize an

interaction of objects, the structure is needed where they are localized.

Opposite case – when some "wandering" objects "meet" other wandering objects, – is interesting, but this discussion will be postponed for a while. The area of programming gives a case when objects, by some way or otherwise, are already packed by in the environment. Thus a central concept under development is namely the environment which is understood as an environment for computations. Environment is equipped with the programming system, but not wise versa.

Other circumstance is that an object interacts not with all environment at once, but with its partition – that which will appear "in an area of action" of the object.

Prestructure. An applicative prestructure is used for packing objects. Two aspects of an object – its redex (reducible expression) and the contract, – reveal in it. In other words, the prestructure gives a representation of computation both in terms of a reduction – transition from redex to the contract, – and in terms of expansion – transition from the contract to redex.

The principle of interaction gives some *non-symmetry*: there is an object-initiator of action and there is an object-recipient of action. Influence of one object on another is stepwise: it is carried out, if and only if objects are located immediately *beside*. The arrangement happens of two kinds: beside and not beside (distant), and in the second case the objects do not interact. In case of an arrangement beside, the objects immediately enter in interaction. The new object, as a result of interaction, arises and begins its existence – result of acting, or applying of the first object to the second. Now, if there will be an object located beside thus newly born object, the new act of interaction begins where are two distinct cases.

In the first of them newly generated object captures the existing one, which has appeared beside and acts on it.

In the second case newly generated object is captured by the existing one which affects this object.

In any of these cases the new object arises and begins its existence and this object is considered as a result of such non-symmetrical interaction of two objects-parents. It settles in prestructure on the equal rights with other objects. In particular, this means the following: as soon as the new object-result is generated, it is possible to speak about the new act of interaction.

Thus, the inhabitants of prestructure participate in interaction which evolves by a principle of a dominoe. The following circumstance is important: either there are *initial* atomic objects, or there are *derived* non-atomic objects, each having exactly two ancestors-parents. A question still open where are the initial objects from, but this discussion will be postponed for a while.

6.2 Interaction

The object can be reveled in interaction with other objects if it participates in application. In this case it can show arity, equal to 0 (constant object) or distinct of zero. For simplicity we shall consider a case when the object shows arity, equal to 1 (unary function).

As interaction is carried out through the intermediary – environment, – then some metaoperators will be required. For a while, we shall be limited by two metaoperators: Λ – currying and $\|\cdot\|\cdot$ – evaluation map. For any object M we shall check up, whether it can show arity 1 in the environment i. To obtain this we write down

$$\|M\|i\, d_0,$$

which represents a value of object M in the environment i. If value of object M shows arity 1 then there is a construction of value of object in the environment

$$\Lambda\|M'\|i\, d_0,$$

where M' is the same as object M everywhere, except for a variable to which we should assign the value d_0: instead of this variable, the number of de Bruijn $\bar{0}$ is written as a prototype of a pointer to d_0 in environment i'. Environment i' is the same as environment i everywhere, except for an image of this substitutional variable, which is now assigned d_0:

$$\|M'\|[i, d_0].$$

Actually, it was necessary to create a compound metaoperator

$$\Lambda\|\cdot\|\cdot : \text{object} \times \text{environment} \to \text{value},$$

which is an object generating, setting up the function of arity 2. Really,

$$\Lambda\|M'\|i\, d_0 = \|M'\|\underbrace{[i, d_0]}_{\equiv i'}.$$

For example, if M is an identity transformation I with the characteristic $Id_0 = d_0$ then it is sufficient to assume, that M' is a substitutional variable which is assigned the value d_0 in environment i:

$$\begin{aligned}\|I\|id_0 &\equiv \Lambda\|\bar{0}\|i\, d_0\\ &= \|\bar{0}\|\underbrace{[i, d_0]}_{\equiv i'}\\ &= Snd[i, d_0] = d_0,\end{aligned}$$

as was expected. Here $\Lambda\|\bar{0}\|i$ is an image of object I, obtained as a result of its interaction with environment. This should

be simply a pointer Snd to d_0, located in the modified environment.

Other example. If M is a cancellator K with the characteristic $Kd_1d_0 = d_1$ then it is sufficient to assume, that M' is a substitutional variable which is assigned the value d_1 in environment i:

$$\begin{aligned}
\|K\|id_1d_0 &\equiv \Lambda(\Lambda\|\bar{1}\|)i\,d_1d_0 \\
&= \Lambda\|\bar{1}\|\underbrace{[i,d_1]}_{i'}d_0 \\
&= \|\bar{1}\|\underbrace{[[i,d_1],d_0]}_{i''} \\
&= (Snd\circ Fst)i'' \\
&= Snd\,i' = d_1,
\end{aligned}$$

as corresponds to the characteristics.

And one more example. If M is the allocator S with the characteristic $Sd_2d_1d_0 = d_2d_0(d_1d_0)$, then

$$\begin{aligned}
\|S\|id_2d_1d_0 &\equiv \Lambda(\Lambda(\Lambda\|\bar{2}\bar{0}(\bar{1}\bar{0})\|))id_2d_1d_0 \\
&= \Lambda(\Lambda\|\bar{2}\bar{0}(\bar{1}\bar{0})\|)\underbrace{[i,d_2]}_{i'}d_1d_0 \\
&= \Lambda\|\bar{2}\bar{0}(\bar{1}\bar{0})\|\underbrace{[[i,d_2],d_1]}_{i''}d_0 \\
&= \|\bar{2}\bar{0}(\bar{1}\bar{0})\|\underbrace{[[[i,d_2],d_1],d_0]}_{i'''} \\
&= \|\bar{2}\|i'''(\|\bar{0}\|i''')(\|\bar{1}\|i'''(\|\bar{0}\|i''')) \\
&= Snd\circ Fst\circ Fst\;i'''(Snd\;i''')(Snd\circ \\
&\quad \circ Fst\;i'''(Snd\;i''')) \\
&= Snd\circ Fst\;i''d_0(Snd\;i''d_0) \\
&= Snd\;i'd_0(d_1d_0) = d_2d_0(d_1d_0),
\end{aligned}$$

as was expected.

7 The principles of computing

Setting up the principles of computing, it is necessary to accept the assumptions, and as it appears, some of the fundamental premises, probably, by virtue of their seeming simplicity and deceptive self-evidence, escape attention of the researchers.

One of them and, possibly, core premise, is in acceptance of applicative structure as an environment of embodiment of computing. It means, that some entities/objects are acting, or *applying* to others. For example, the object-function *is applied* to object-argument, and the result of this application is considered as a *value of* the given function of the given argument. All of this looks quite obvious, but is almost never formulated explicitly, resulting in various displacement of accents.

Another – and again all known, – assumption can be dropped down to a simple formulation of operating with identifiers: given that is considered as an *identifier*, and for this *relatively the environment* is *constructing* that will be considered as a *value*. This process of constructing is considered as a computation (in a sense of evaluation), and computing develops technologies for realization these constructions. Thus, the relation between the identifier and its value is parameterized by an environment.

In the environment not all the objects are isolated, an interaction of the objects is the most interesting, but it proves through application. This is a structural metaoperation which operates an interaction of objects and it would be undesirable, that during performance of computation of value of the identifier its own properties have been changed. Hence applying will be considered as invariant relatively evaluation, and this property needs obvious characterization by

acceptance of the general principle of evaluating the application (V. E.Wolfengagen, [22]).

Principle of evaluating the application. The base premise is as follows:

evaluation of application is the application of evaluations.

The formulation above needs augmentation because of evaluation can be executed both under fixed and unfixed environment. In the first case when the environment i is assumed as fixed suppose:

$$\|MN\|i = (\|M\|i)(\|N\|i)$$

for arbitrary objects M, N. Then by purely formal reasons, following from the general properties of computations,

$$\begin{aligned}(\|M\|i)(\|N\|i) &= (\lambda r.r\|M\|\,\|N\|)Si \\ &= CIS(\lambda r.r\|M\|\,\|N\|) \\ &\equiv \mathcal{S}[\|M\|, \|N\|]i\end{aligned}$$

for $S \equiv \lambda xyz.xz(yz)$, $C \equiv \lambda xyz.xzy$, $I \equiv \lambda x.x$, $\mathcal{S} = CIS$ and ordered pair $[x, y] \equiv \lambda r.rxy$. Hence, the following rule

(rule 1) $$\|MN\| = \mathcal{S}[\|M\|, \|N\|]$$

can be formulated, which is derivable in $\lambda\eta\xi$-calculus.

Principle of evaluating the ordered pair. The main premise is formulated as the equality

evaluation of ordered pair is the ordered pair of evaluations.

Formally this equation is rewritten as

$$\|[M, N]\|i = [\|M\|i, \|N\|i]$$

for any terms M, N and assignment i. By the postulates (η), (ξ) and (τ) of λ-calculus, the following rule is derivable from the main principle:

(rule 2) $$\|[M, N]\| =< \|M\|, \|N\| >,$$

where $< f, g >\equiv \lambda t.[ft, gt]$ for arbitrary f, g. Both the principles above and derived rules make it feasible to obtain and ground the standard semantic features of computational models. As the most important for the means of conceptualization we indicate two corollaries which correspond the evaluation of λ-expressions and applications as is written below.

Evaluation of λ-expression. Assume the application $(\lambda.M)\bar{d}$, where M, $\bar{d}$ are any terms, and $(\lambda.M)$ denote the abstraction of (some) variable. Then the following consequence of equalities is valid:

$$\begin{aligned}
\|(\lambda.M)\bar{d}\|i &= (\|(\lambda.M)\|i)(\|\bar{d}\|i) && \text{(by principle 1)}\\
&= (\|\lambda.M\|i)d && \text{(assuming } (\|\bar{d}\|i) = d)\\
&= \Lambda\|M\|id && \text{(by definition of } \Lambda)\\
&= \|M\|\,[i, d] && \text{(by } \Lambda h = \lambda xy.h[x, y]).
\end{aligned}$$

Hence, the rule:

(rule 3) $$\|(\lambda.M)\bar{d}\|i = \|M\|\,[i, d].$$

is valid as well. In the following we will accept that this rule determines a generating function: evaluation of M "generates" the replacements by d matching M.

Second way of evaluating the application. In evaluation of application the definition of applicator ε:

$$\begin{aligned} \|MN\|i &= (\|M\|i)(\|N\|i) && \text{(by principle 1)} \\ &= \varepsilon[\|M\|i, \|N\|i] && \text{(by definition of } \varepsilon) \\ &= (\varepsilon\circ < \|M\|, \|N\| >)i && \text{(by definition of } < \cdot, \cdot >) \end{aligned}$$

is used. This sequence of equalities leads to the rule

$$\text{(rule 4)} \qquad \|MN\| = \varepsilon\circ < \|M\|, \|N\| > .$$

At last, we indicate the case of evaluation of individual constants:

$$\|c\|i = c,$$

which (in $\lambda\eta\xi$-calculus) results in the rule

$$\text{(rule 5)} \qquad \|c\| = \lambda i.c.$$

This means that individual constants are independent on the particular assignment, i.e. they are statical.

List of rules. For reference, a complete list of the derived rules is given below:

$$\begin{aligned} &\text{(rule 1)} && \|MN\| = \mathcal{S}[\|M\|, \|N\|] \\ &\text{(rule 2)} && \|[M, N]\| = < \|M\|, \|N\| > \\ &\text{(rule 3)} && \|(\lambda.M)\bar{d}\|i = \|M\|\,[i, d] \\ &\text{(rule 4)} && \|MN\| = \varepsilon\circ < \|M\|, \|N\| > \\ &\text{(rule 5)} && \|c\| = \lambda i.c. \end{aligned}$$

8 Interaction of objects

8.1 Unfixed number of argument places

First of all it would be desirable do not link the interaction of objects to traditional mathematical representations. At least, not to do this at once and without any visible necessity. The mathematical intuition prompts, that if something acts on another it is necessary to consider the first as a function, and the second – as an argument. Under these conditions the result of interaction is considered as value which, in turn, is an object.

But if there is a function, it is characterized by its arity – by number of its arguments or, more precisely, argument places. In the usual mathematics the number of arguments of a function is known in advance, but in case of acting of one object on others it is not known in advance, on how many objects it can act. If the object is considered as a function, then its number of argument places appears in advance unfixed, and the object reveals its arity in interactions. Whether such mathematical functions are known? As it appears they are only the combinators.

8.2 Object and its sphere of action

Let there is some set of objects arranged in a structure. We shall start from the point that not any object will cooperate with everyone. If we speak, that the object a acts on object b, we have in mind that a is applied to object b.

From Fig. 1 it is clear that in the sphere of action of object X there are both the objects a, and b, but a does not act on b. In a mathematical notation it is written down below:

$$X..\ ab..\ \equiv (..(((X..)a)b)..).$$

From Fig. 2 it is clear that in a sphere of action of object X

Fig. 1. Object X affects the objects a and b, where a does not affect b.

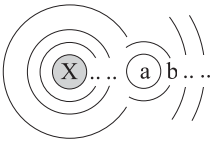


Fig. 2. Object X acts on the object (ab), where a acts on b.

there is an object a, which acts on b, so that X acts on a result of action of object a on object b. Notationally this is written down as follows:

$$X..(ab)..\ \equiv (..((X..)(ab))..).$$

Parentheses in which objects a and b are concluded, are *essential*, and they cannot be omitted.

9 System of primary objects

Possibly, it is necessary to make some assumptions. They will concern presence of some set of primary objects – in fact, it is necessary to have in stock actual, actually existing objects. At the same time we shall make an attempt to conduct discussion of relations between objects, whenever possible, without those assumptions, in particular, concerning the existence of objects which can be avoided.

First, an ability *to generate* any object b is required. Let it will be always accompanied by occurrence and application

of a constant object K. Other version of this reason can look differently: it would be desirable to formulate the statement, that some object a does *not* depend on *environments* b. It means also, that purely by syntax the object K incurs function of *quoting* a in an environment or in a context b. Application of K also means *encapsulation* of object a within environment b. Each of these explanatory systems can be used depending on a context in which studying of objects is conducted.

On the other hand, there is also a symmetric opportunity of *elimination*, or the *termination of existence* – cancelation out, – any object b. The termination of existence of object b is caused by the termination of existence of a constant object K, directly acting on some object a, and the object a remains and continues its existence.

From Fig. 3 it is possible to understand behaviour of object-

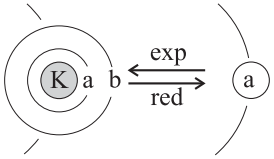


Fig. 3. Characteristic of object K: $a = Kab$.

cancellator K. On expansion of any object a the object K is generated, starting its existence, and object a appears directly in a sphere of its action. In additon the object b is generated which gets in a sphere of action of a result of interaction of object K with object a. Upon reduction the cancellator K eliminates object b which ends its existence, but this instance of K also does not exist any more.

At the same time it can be demanded to *eliminate clones* of object, leaving only its single instance. Certainly, the symmetric operation of distribution of actions on various instantiations of any object c, carrying out their *cloning* is expected as well. We shall try to carry this out as follows: on elimination of a clone of object c the constant object S, of a special kind, will be generated, and on cloning c – to the contrary, one of instantiations of object S will be enforced to end its existence.

All of this means, that various instances of object will be indiscernible in applicative environment. This idea of indiscernibility of a clone of object is presented at the scheme of computation which determines a behaviour of the allocator S: $ac(bc) = Sabc$. Apparently from Fig. 4, that object S –

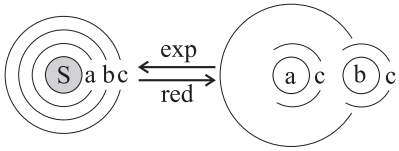


Fig. 4. Characteristic of object S: $ac(bc) = Sabc$.

in applicative environment, – is *applied* to objects a, b and c, capturing them. This means also that the arity of initial primary object-combinator S equals 3. Its action directly does not extend on the other objects in the environment. But all the construction of $Sabc$ is *reduced* to $ac(bc)$, the object c is cloned, and computation is distributed: one instance of c directly occurs in the sphere of applying of a, and its second instance – in the sphere of b. Computation is distributed, but the result obtained from interaction b with c, occurs in a sphere of action of the result obtained from interaction a with c. The most essential,

that at generation of a new object S: $ac(bc) = Sabc$ the clone of object c stops its existence, and object S starts its existence, becoming an inhabitant of applicative environment. This is an essence of *S-expansion*. On the other hand, at acting of object S on kept by it in the environment objects a, b and c, occurs the cloning of c, this clone begins its existence in the environment, but at the same time S stops its existence. This characterizes *S-reduction*.

10 System of derived objects

Now the way of "detecting" the objects with predefined characteristics in applicative environment is, in general, clear.

Process of detection of a new object appears rather constructive: it is necessary to show a construction built from already found out, old objects. Thus, new objects appear *derivatives* while gradually explicate a representation about the *initial* objects, all or part of which appears the *primary* ones.

Let's consider, by example, the synthesis of a new object with predefined characteristic. Let in Fig. 5 the characteristics of such an object is as follows: this is the combinator-permutator C, carrying out rearrangement by places of objects b and c.

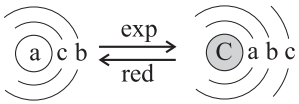


Fig. 5. Synthesis of object C with predefined combinatory characteristic: $Cabc = acb$.

A synthesis of C is carrying out as follows. Starting with object acb, which by its structure is a result of applying to b the result of applying a to c.

First, we shall execute cloning of c. For this purpose we shall make K-expansion on the basis of object b, generating the second instance of object c, which is possible to see in Fig. 6. Second, execute S-expansion on the basis of object $ac(Kbc)$ in

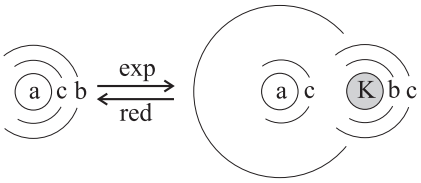


Fig. 6. K-expansion and generation the second instance of object c: $acb = ac(Kbc)$.

accordance with Fig. 7. During its execution the second instance

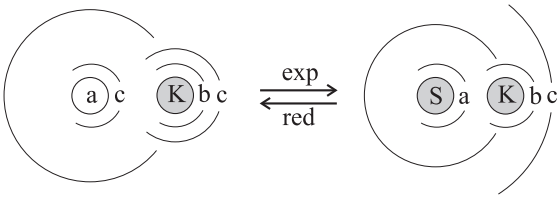


Fig. 7. S-expansion and elimination of the second instance of object c with permutation of object Kb: $ac(Kbc) = Sa(Kb)c$.

of object c is eliminated, but the allocator S is generated.

Third, execute B-expansion according Fig. 8.

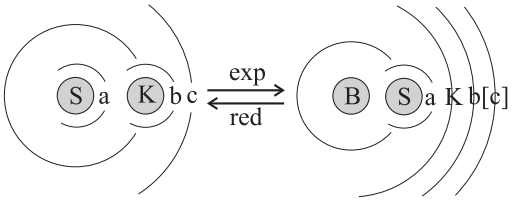


Fig. 8. B-expansion and elimination of a composition of objects Sa and K: $Sa(Kb)c = B(Sa)Kbc$.

Fourth, execute B-expansion, which eliminates the composition of objects B and S. The same time execute the cloning of object a, using K-expansion based on object K. The new instances of objects K and a are generated, starting up the existence in an environment. This transformation is done in accordance with Fig. 9.

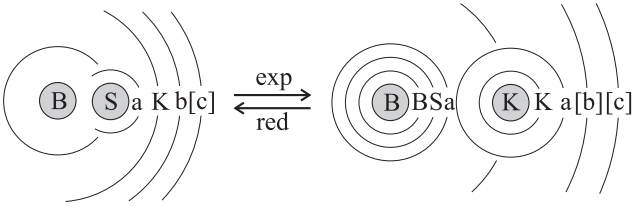


Fig. 9. B-expansion and elimination of composition of objects B and S, carried out together with K-expansion based on K with generation the clone of object a: $B(Sa)Kbc = BBSa(KKa)bc$.

And, at last, fifth, execute S-expansion, which eliminates the distribution of computations $BBSa$ and KKa together with clone of object a. The second instance of object a cancels out

the existence, but the object S – starts up the existence in environment. This is represented in Fig. 10. Now the desired

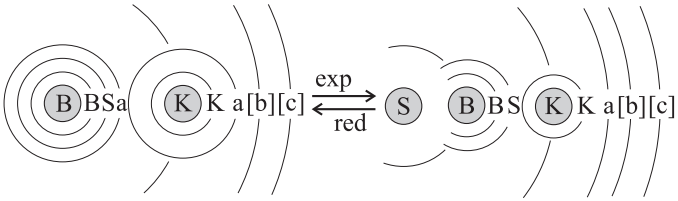


Fig. 10. S-expansion and elimination of a clone of the object a: $BBSa(KKa)bc = S(BBS)(KK)abc$.

order of objects a, b and c is achieved in environment, i.e. objects c и b have changed their places.

In this process the combinator B with the characteristic $Babc = a(bc)$ is used. It seems clear that it can be synthesized as well, using the only combinators K and S. In Fig. 11 a

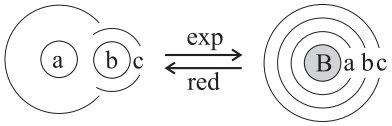


Fig. 11. Characteristics of the combinator B.

characteristic of combinator B is shown.

We will show that combinator B can be obtained by combining K and S. First of all let's fix the object $a(bc)$. The idea is to get the object c free of immediate action from the object b. Mathematically this means the need to omit

parentheses. To get this, the distribution of computations will be synthesized, generating an additional instance of object c, that is reached by occurrence of an instance of combinator K. Write down symbolically:

$$a(bc) = Kac(bc).$$

Further, we eliminate one of instances of object c, that needs the occurrence of an instance of combinator S, but, in passing, remained instance of c is get out of dependence on b:

$$Kac(bc) = S(Ka)bc.$$

Similarly we shall get out now object a of dependence on object K. It is necessary to do this in two steps. First we shall generate the second instance of a, having distributed computation and having generated an instance of combinator K:

$$S(Ka)bc = KSa(Ka)bc.$$

In Fig. 12 all the derivation chain is shown as

$$a(bc) \lhd Kac(bc) \lhd S(Ka)bc \lhd KSa(Ka)bc.$$

Now one of two instances of object a will be eliminated, for which it is necessary to generate the object S:

$$KSa(Ka)bc = S(KS)Kabc.$$

The target object is synthesized, it remains to assume only that

$$S(KS)Kabc \equiv Babc.$$

In Fig. 13 a continuation of derivation is shown, starting with $KSa(Ka)bc$:

$$KSa(Ka)bc \lhd \underbrace{S(KS)K}_{\equiv B} abc \equiv Babc.$$

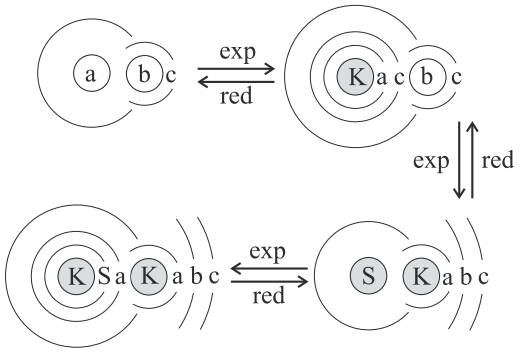


Fig. 12. Derivation of combinator B: $a(bc) \lhd Kac(bc) \lhd S(Ka)bc \lhd KSa(Ka)bc$.

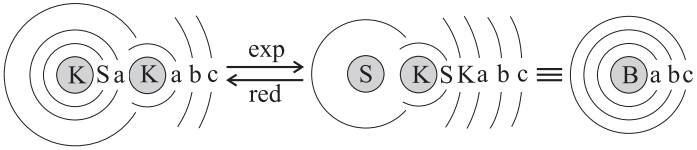


Fig. 13. Derivation of combinator B: $KSa(Ka)bc \lhd S(KS)Kabc \equiv Babc$.

Thus, the full chain of a derivation looks like

$$a(bc) \lhd Kac(bc) \lhd S(Ka)bc \lhd KSa(Ka)bc \lhd \underbrace{S(KS)K}_{\equiv B} abc \equiv$$

$$\equiv Babc.$$

By the same way it is possible to synthesize identity combinator I with the characteristic $Ia = a$, that is shown in Fig. 14. This

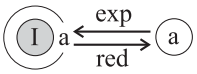


Fig. 14. Characteristic of combinator I: $Ia = a$.

is a special combinator, under its action any object a does not vary, and speaking more exactly, remains self-identical, passing in itself.

A derivation for combinator I is shown in Fig. 15. We start

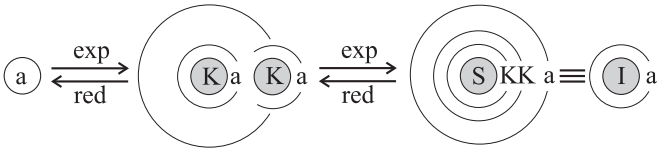


Fig. 15. Derivation of combinator I: $a = Ka(Ka) = SKKa \equiv Ia$.

fixing the object a. We generate object (Ka), but this arises an instance of object K as well, which starts its own existence.

Now it has appeared, that two instances of object a are available which structural arrangement allows to eliminate one of them. But thus the object S, which begins the existence, is generated. Now it has appeared, that the object SKK by its characteristic does not differ from the object I, demanded by the synthesis. So the purpose is reached, and mathematically it can be written down by means of $I \equiv SKK$.

Primary and obtained during the formation of objects, and some of the derived combinators are represented in Table 1.

11 Derivation of combinators

Given such an atomic-and-molecular constructor, it is possible to synthesize the objects with predefined computational properties – the combinatory characteristics.

In the Fig. 16 in a scheme for combinator S the object b

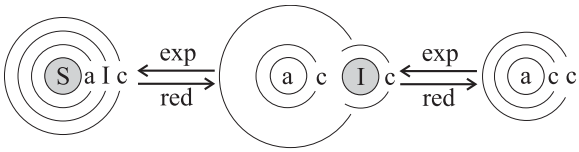
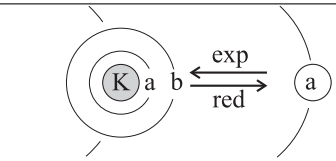
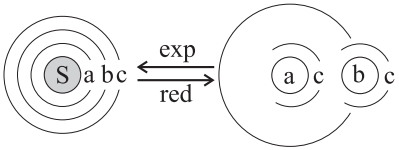
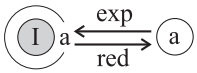
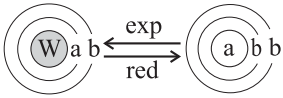
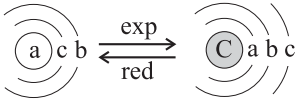
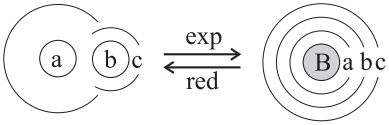
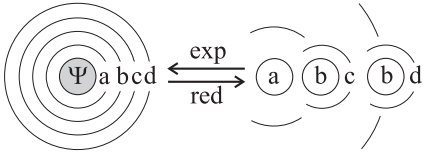
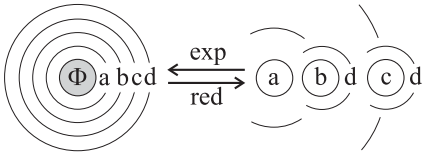


Fig. 16. A particular construction of combinator S for $b \equiv I$.

is replaced by the combinator I. This initiates a reduction in the direction from $SaIc$ to $ac(Ic)$. The last object contains the redex Ic, which is replaced by the contract c, so that all the reduction works as follows:

$$SaIc \rhd ac(Ic) \rhd acc.$$

Table 1. Primary and derived combinators.

Combinatory characteristics	Interaction of objects
$Kab = a$	
$Sabc = ac(bc)$	
$Ia = a$	
$Wab = abb$	
$Cabc = acb$	
$Babc = a(bc)$	
$\Psi abcd = a(bc)(bd)$	
$\Phi abcd = a(bd)(cd)$	

But inside the object $SaIc$ it is necessary to make transformations: this concerns a position of combinator I. Transformation will be executed by expansion:

$$SaIc \lhd \underbrace{CSI}\, ac \equiv Wac, \\ \equiv W$$

but both the chains of synthesizing the object can be merged:

$$Wac \equiv CSIac \rhd SaIc \rhd ac(Ic) \rhd acc.$$

A process of synthesis the object W is represented in Fig. 17. Now the newly synthesized combinator W, which, as

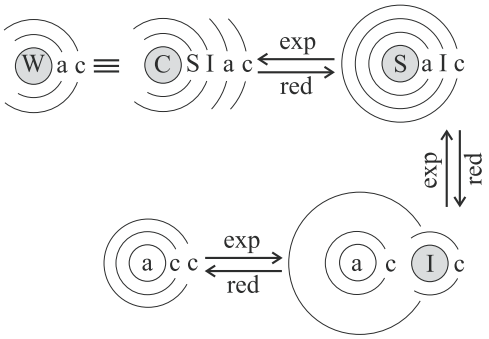


Fig. 17. Synthesis of construction of combinator W with a characteristic $Wac = acc$.

appeared, behaves exactly like CSI, can be added to available combinators. In Fig. 18 a characteristic of combinator W is given.

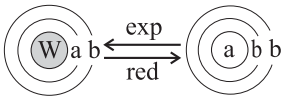


Fig. 18. Characteristic of combinator W.

12 Reduction and expansion of objects

Once again turn back to features of obtaining the combinatory representation of object-permutator C. There are in Fig. 19 the

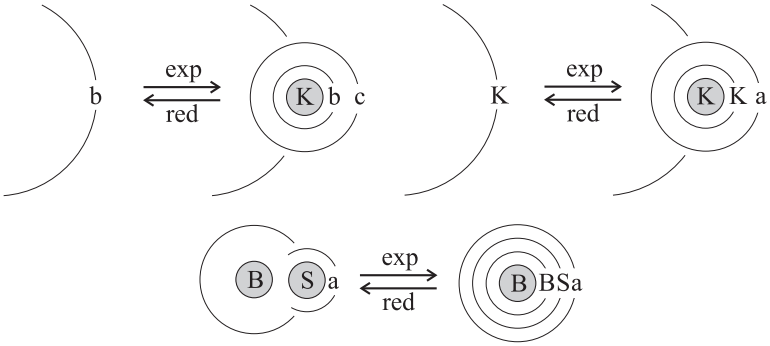


Fig. 19. Synthesis of supplementary objects: $b = Kbc$, $K = KKa$, $B(Sa) = BBSa$.

supplementary objects, which are obtained during a synthesis of target object C with predefined combinatory characteristic

$$Cabc = acb.$$

Everything what is required is to change places of the second and third objects in acb. However, it is necessary to remember

that rearrangement should be made in applicative environment. First of arising ideas consists in trying to find transformation of acb, having executed which it is possible then to apply transformation under the scheme S.

Such an attempt is reflected in Fig. 6 and led to a need of generation the additional instance of an object c – its *clone*, but *not* a copy. In other words, distinct instances of object in applicative environment will be indiscernible. Such an idea of indiscernibility of clone of object is present in the scheme of computation, defining behaviour of the distributor S: $ac(bc) = Sabc$. Apparently from Fig. 4, the object S – in applicative environment, – is *applied* to objects a, b и c, keeping them.

13 Synthesis of an object with the given combinatory characteristic

Let's return to consideration Fig. 5.

It is required to clone object c, and then to eliminate it, having taken advantage of S-expansion.

First, we shall execute cloning of c. For this purpose we shall make K-expansion on the basis of object b, generating the second copy of object c, what is possible to see in Fig. 6.

Second, we shall execute S-expansion according to Fig. 7.

Third, we shall execute B-expansion according to Fig. 8.

Fourth, we shall execute B-expansion which eliminates a composition of objects B and S. At the same time we shall execute cloning of object a, using K-expansion on the basis of object K. New instances of objects K and a are generated, beginning the existence in the environment. This transformation is carried out according to Fig. 9.

And, at last, fifth, let's execute S-expansion which eliminates distribution of computations $BBSa$ and KKa together with a clone of object a. The second copy of object a cancels out its existence, and object S – begins the existence in the environment. It is reflected in Fig. 10.

Now the desirable order of arranging the objects a, b and c in the environment is reached, that is the objects c and b have changed the places. The derived object, which is carrying out such a rearrangement, we shall refer as combinator-*permutator* and we shall denote it by C. From Fig. 10 it follows, that $C \equiv S(BBS)(KK)$. As it has appeared, C exists, and its characteristic is presented in Fig. 5.

14 Infinite constructions

Objects can lead to infinite constructions, and their interaction has chained. Fig. 20 depicts a combinatory characteristic – the

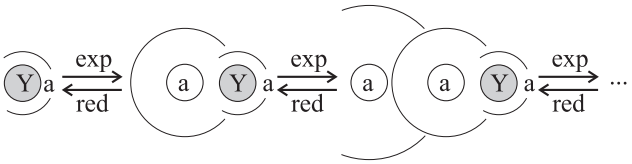


Fig. 20. Synthesis for an object a the fixed point: $Ya = a(Ya) = a(a(Ya)) = \ldots$.

characteristic equality that needs to be added to a structure of combinators to give a right to existence for combinator Y:

$$Ya = a(Ya),$$

and this is a paradoxal combinator of H. B. Curry[2]. Combinator Y makes more vivid rather uniform structure of objects and combinators, giving rise to opportunity of organizing non-trivial cyclic computations.

15 The plurality of the worlds of combinators

Attempts to answer the question, what – actually, – are the *constants* often look boring enough, tiresome and burdened by a set of more or less essential details. For the sake of justice we shall note, that more often, from the resulted details their most part appears not significant or, at least, not spilling greater light. Possibly, in an area of computing, it will be possible to receive missing details, and to look under other corner of sight at the old ones. At least, the idea of constancy appears *relative*, i.e. it is useful for considering not in a general universality, and remaining within the limits of this or that system of computing. We shall try to understand by an example, where it can appear useful.

The set of objects together with their structuring combinators looks rather homogeneous, even together with opportunities for generation of cyclic constructions using the combinator Y. It seems, that in similar structure there is no place to any systems of objects with interesting and practically significant behaviour. But this is not so.

Let's try to establish, whether will there be among objects such an object V which is characterized by the distribution relatively an application as follows:

$$V(ab)\rho = Va\rho(Vb\rho).$$

[2] As known the object Y has a combinatory representation: $Y \equiv WS(BWB)$.

Thus, all that is obtained is the structure of objects which is enriched by this equality, and its action is illustrated in Fig. 21.

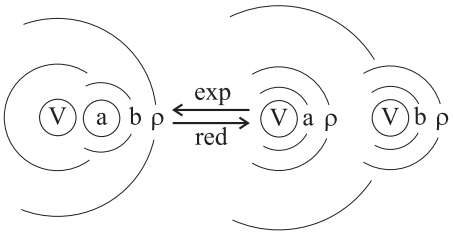


Fig. 21. Characteristic equality for object V: $V(ab)\rho = Va\rho(Vb\rho)$.

The kind of the left part of this equality suggests, that there is a composition of the objects-maps V and a which is applied to object-argument b, and the result of this application, in turn, is applied to object ρ, playing a role of environment:

$$V(ab)\rho = ((V \circ a)(b))(\rho) \equiv (V \circ a)b\rho \equiv BVab\rho.$$

This mathematical idea is reflected in Fig. 22.

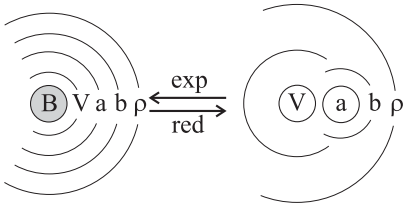


Fig. 22. Canonical representation of the left side of the characteristic equality of object V: $V(ab)\rho = BVab\rho$.

Let's borrow now in the right part of the equality describing behaviour of object V, and we shall try to receive its initial representation. Fortunately, the most part of combinational work on distribution of computations among objects incur combinators Ψ and Φ, and corresponding reduction looks laconically enough:

$$Va\rho(Vb\rho) = I(Va\rho)(Vb\rho) = \Phi I(Va)(Vb)\rho = \Psi(\Phi I)Vab\rho$$

Corresponding transformations are depicted in Fig. 23.

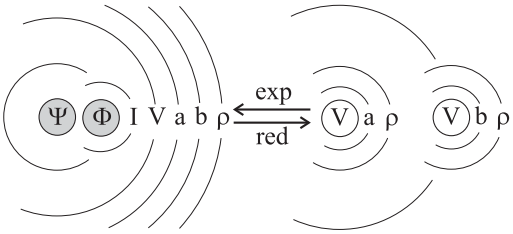


Fig. 23. Canonical representation of the right side of the characteristic equality of object V: $Va\rho(Vb\rho) = \Psi(\Phi I)Vab\rho$.

For the purposes of the further ordering we shall merge the executed transformations:

$$BVab\rho = V(ab)\rho = Va\rho(Vb\rho) = \Psi(\Phi I)Vab\rho$$

The frame part of a chain of the derivation is presented in Fig. 24.

What follows from the transformations above? It is possible to receive several consequences.

First of all, it is easy to be convinced, that for $V \equiv K$ the resulted construction works, so the object V does *exist*.

At the same time it is possible to write down mathematically laconic sentence fixing the basically obtained result:

$$\exists V.\forall a, b, \rho.BVab\rho = \Psi(\Phi I)Vab\rho.$$

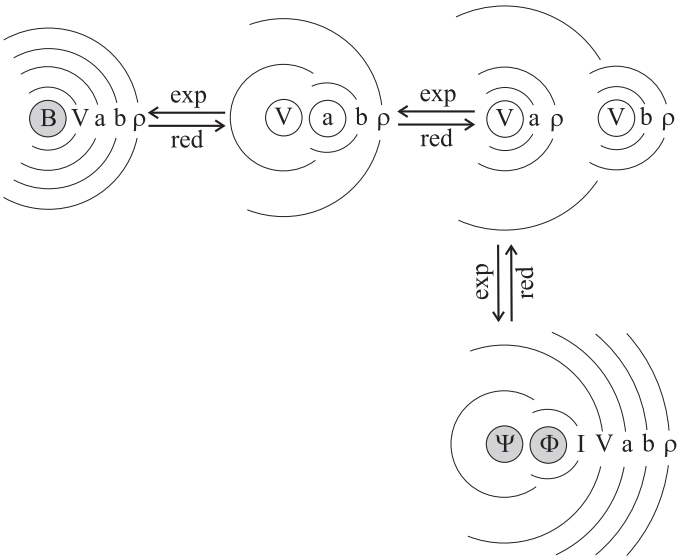


Fig. 24. Resulting characteristic equality for object V: $BVab\rho = V(ab)\rho = Va\rho(Vb\rho) = \Psi(\Phi I)Vab\rho$.

This means that there is an object V such that for any objects a, b, ρ the equality

$$B = \Psi(\Phi I),$$

is satisfied and characterizes the object V. Note that the object ρ can be taken as an *environment* in the sense which is included in this term in a theory of programming languages (see. [20]; [22], Ch. 12). The stated reasons lead to a construction resulted in Fig. 25.

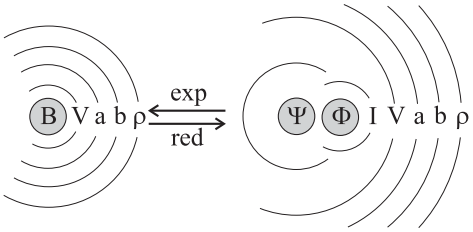


Fig. 25. Combinatory characteristic generating the object V: $\exists V.\forall a, b, \rho.BVab\rho = \Psi(\Phi I)Vab\rho$.

Acknowledgements

Undoubtedly, the discussion of computing at the Conference «Applicative computing systems» – ACS-2008, that was held in April-May, 2008 at the Institute «JurInfoR-MGU», has served as effective stimulus for some arranging the stated reasons.

Conclusions

Naturally, all of this is a fixing of some key ideas, but development of plurality of the worlds of combinators seems worthy and justifying efforts to preparation of a preliminary textual material. The part from them has appeared included in the Proceedings of ACS-2008, see URL `http://jurinfor.exponenta.ru/ACS2008`. Other part is supposed to a consecutive statement.

References

1. Barendregt H., Wiedijk F. *The Challenge of Computer Mathematics*. – Transactions of the Royal Society, Vol. 363, №1835, 2005. – pp. 2351-2375. `ftp://ftp.cs.ru.nl/pub/CompMath.Found/Barendregt-Wiedijk.pdf`

2. Bell G., Dourish P. *Yesterday's tomorrows: notes on ubiquitous computing's dominant vision*. – Personal Ubiquitous Comput., Vol. 11, №2, 2007. – pp. 133–143.
DOI http://dx.doi.org/10.1007/s00779-006-0071-x.
3. Berners-Lee T., Hall W., Hendler J., O'Hara K., Shadbolt N., and Weitzner D. *A framework for Web science.* – Foundations and Trends in Web Science, Vol. 1, Issue 1, 2006. – pp. 1-130.
http://www.nowpublishers.com/web/
4. Cardelli L., Davies R. *Service combinators for Web computing.* – HP Labs Technical Reports SRC-RR-148, June 1, 1997. – 15 p.
http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-148.html
5. Carpenter B. *The Internet Engineering Task Force: Overview, Activities, Priorities.* – ISOC BoT, 2006-02-10, 2006.
http://www.isoc.org/isoc/general/trustees/docs/Feb2006/IETF-BoT-20060210.pdf
6. Curry H. B. *Functionality in combinatory logic.* – Proc. National Academy of Sciences of the USA, Vol. 20, 1934. – pp. 584–590.
7. de Bruijn N. G. *Lambda-calculus notations with nameless dummies: a tool for automatic formula manipulation.* – Indag. Math. 1972, №34, pp. 381-392.
8. de Bruijn N. G. *A survey of the project Automath*. – In: To H.B. Curry: Essays in combinatory logic, lambda calculus and formalism, Academic Press, 1980. – pp. 579-606.
9. Denning P. J. *Computing is a natural science.* – Commun. ACM, Vol. 50, №7, 2007. – pp. 13–18.
DOI http://doi.acm.org/10.1145/1272516.1272529
10. Hindley J. R., Lercher B., Seldin J. P. *Introduction to Combinatory Logic.* – London: Cambridge University Press, 1972.
11. Kennedy A. *Functional Pearls: Pickler Combinators.* – Journal of Functional Programming, special issue on Functional Pearls, Vol. 14, №6, Cambridge University Press, Nov 2004. – pp. 727-739,
12. MacLennan B. J. *Molecular Combinator Reference Manual.* – UPIM Report 2, Technical Report UT-CS-02-489, Department of Computer Science, University of Tennessee, Knoxville, 2002. – 16 p.
http://www.cs.utk.edu/~mclennan/UPIM/CombRef.pdf
13. MacLennan B. J. *Combinatory Logic for Autonomous Molecular Computation.* – Preprint of paper invited for Information Sciences, 2003.
http://www.cs.utk.edu/~mclennan/UPIM/CLAMC-IS.pdf
14. MacLennan B. J. *Molecular Combinatory Computing for Nanostructure Synthesis and Control.* – IEEE Nano 2003, San Francisco, August 12-14, 2003.
http://www.cs.utk.edu/~mclennan/UPIM/MacLennan-MCCNSC.pdf
15. Schönfinkel M.I. *Über die Bausteine der mathematischen Logik*. Math. Annalen 92, 1924. – pp. 305–316.

16. Scott D. S. *The lattice of flow diagrams.* – Lecture Notes in Mathematics, 188, Symposium on Semantics of Algorithmic Languages. – Berlin, Heidelberg, New York: Springer-Verlag, 1971, pp. 311-372.
17. Scott D. S. *Relating theories of the lambda calculus.* – Hindley J., Seldin J. (eds.) To H.B.Curry: Essays on combinatory logic, lambda calculus and formalism.- N.Y.& L.: Academic Press, 1980, pp. 403-450.
18. Seldin J. P. *The Logic of Curry and Church*. – In: (Dov Gabbay and John Woods, eds.) *Handbook of the History of Logic*, Vol. 5, Elsevier, 2006. `http://people.uleth.ca/~jonathan.seldin/CCL.pdf`
19. Selinger P. and Valiron B. *A lambda calculus for quantum computation with classical control.* – Mathematical Structures in Computer Science, 16(3), 2006. – pp. 527-552.
20. Вольфенгаген В. Э. *Конструкции языков программирования. Приемы описания.* – М.: АО «Центр ЮрИнфоР», 2001. – 276 с.
    Издание поддержано грантом РФФИ, проект 01-01-14068-д.
21. Вольфенгаген В. Э. *Комбинаторная логика в программировании. Вычисления с объектами в примерах и задачах.* – М.: МИФИ, 1994. – 204 с.; 2-е изд., М.: АО ≪Центр ЮрИнфоР≫, 2003. – 336 с.
22. Вольфенгаген В.Э. *Методы и средства вычислений с объектами. Аппликативные вычислительные системы*. – М.: JurInfoR Ltd., АО «Центр ЮрИнфоР», 2004. – xvi+789 с.
    Издание поддержано грантом РФФИ, проект 03-01-14055-д.
23. Исмаилова Л. Ю. *Логика объектов*. – В кн. [22], с. 613-630.
24. Косиков С. В. *Логика функциональности*. – В кн. [22], с. 595-612.
25. Косиков С. В. *Информационные системы: категорный подход*. – Под ред. Л. Ю. Исмаиловой. – М.: «ЮрИнфоР-Пресс®», 2005. – 96 с.
26. Шаумян С. К. *Аппликативная грамматика как семантическая теория естественных языков.* – М.: Наука, 1974. – 204 с.

Index

Part 2

Educational and methodical complex of corresponding discipline and ready-made solutions

Multimedia courses

The system of multimedia courses is intended to cover the most complicated and important directions of modern computing, computer science and information technologies.

♦ ♦ ♦

Combinatory logic

Course content. This course was delivered in NRNU MEPhI and MIPT including 13 two hours lectures and reflects the modern representations of fundamental basics of computing. The main attention is paid to development the learners' intuition which is sufficient to understand the granularity of computations, its organizing using the block structure of objects, the ability to construe the embedded computational systems. In many cases the accent is given on development the suitable embedded computational systems.

The course is equipped with the detailed textbooks with a large amount of examples and tasks with the analysis of solutions. An organization of books corresponds to the multimedia course and allows to study the material in a variant of «for the first reading» and does not demand the learner's

preliminary background. In addition, as a result of studying the course, a student acquires the knowledge of the primary problems of computing which are at the frontier of modern computer science.

♦ ♦ ♦

Wolfengagen V. E. *Applicative Computational Technologies. Ready-made Solutions for Engineer, Lecturer, Post-graduate and Graduate Student.* Edited by L. Yu. Ismailova. — Moscow: «JurInfoR», 2009. — 64 p. (in Russian)

Summary. This work reflects the problems of computing using the advanced and contemporary mathematical means. Material is subdivided in two parts and is approved in practice of teaching at NRNU «MEPhI», MIPT and several other educational institutions of the country. The 1st part represents the outlook of computations, which is achieved by the adoption of the atomistic doctrine for the selected reference system of primary objects. This part is intended for the advanced learners of discrete mathematics (DM) and fundamental basics of information technologies (FBIT), it is suitable for developing the intuition, which helps to the successful navigation across the world with apparent abundance and variety of the newly developed and ready-made IT-solutions. The 2nd part gives the educational and methodical complex of corresponding discipline (EMCD), equipped with base textbooks, complemented by a series of monographs and by media and environments for studying the computations with the objects.

Material is intended for the instructor, the graduate student and the student of IT-specialties and is the ready-made solution

for the universities and the system of training and advancing the qualification of specialists.

♦ ♦ ♦

Wolfengagen V. E. *The Functional Programming Paradigm.* Edited by L.Yu. Ismailova. — Moscow: «Center JurInfoR», 2010. — 80 p. (in Russian)

Summary. This work covers the main directions of evolving functional programming ideas and technologies. This material is approved in practice of NRNU MEPhI, MIPT and several other educational centers of the Russian Federation.

Its 1st part represents an outlook of trends in usage the pure functions in programming and is suitable both for advanced learners and beginners in Computing and Information Technologies.

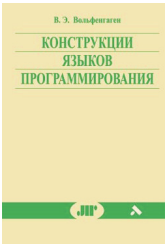
The functional programming paradigm shown its great importance and significantly influenced many branches of computing and computer science. This is an evolving roadmap having a perspective of growth in software engineering. It's happened that program correctness is much easier to be proved in case it is written in functional language. With evolution of tools for supporting the proof this process seems to become even easier. The functional programs can be treated as executable specifications which themselves are already prototypes, i.e. they do not need any additional efforts for their preparing. The transformations of functional programs are greatly simplified because of an algebraic origin of the functions. Their usage opens the abilities for developing the innovative machinery of the code optimization which grows up the efficiency both of sequential and concurrent architectures.

The amount of cases when an adoption of functional approach for solving the real world tasks gives the obvious advantages tends to growing up.

The 2nd part gives some supply of environments for educational and methodical complex of corresponding discipline (EMCD).

Material is intended for the instructor, postgraduate and graduate students of IT-specialties.

♦ ♦ ♦



Wolfengagen V. E. *The constructions of programming languages. The techniques of description.* — Moscow: «Center JurInfoR» Ltd., 2001. — 276 p. (in Russian)

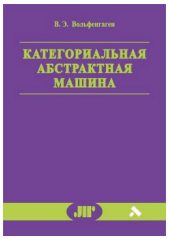
Summary. This book covers the basics of development, implementation and application of constructions both of imperative and functional programming languages. An essential attention is paid to application of denotation semantics which allows in a complete degree extracting the advantages of an object-oriented approach and this, in turn, as a final score allows developing the target computational model of purely functional kind.

The material is supplemented by the examples with detailed explanations which are carefully commented assisting to study the implementation of constructions from various languages.

This material can be used as a textbook or a guide. It will be useful both for students and postgraduates, and for professionals in computer science, information technologies and programming.

RFBR's project 01-01-14068-д.

♦ ♦ ♦



Wolfengagen V. E. *Categorical Abstract Machine. Conspectus: Introduction to Computations.* — 2nd ed. — Moscow: «Center JurInfoR», 2002. — 96 p. (in Russian)

Summary. This book contains the basics for computation models in Computer Science. The core topics both of lambda-calculus and combinatory calculi are covered.

The main goals are to provide formal tools to assess meaning of programming constructs in both a language-independent and a machine independent way including the code compiling and generation, its optimization and runtime considerations. This is done using the categorical abstract machine - relatively new direction in studying and understanding the programs and computations. The material is equipped with serial examples of growing up complexity.

This book is recommended to the undergraduate and postgraduate students in Computer Science, Programming Languages, Information Technologies, and Discrete Mathematics.

♦ ♦ ♦

Wolfengagen V. E. *Combinatory logic in programming. Computations with objects through examples and exercises.* — 2-nd ed. — Moscow: «Center JurInfoR», 2003. — VI+336 p. (in Russian)

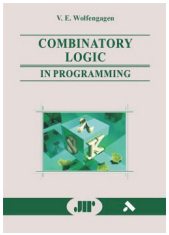
Summary. The book is intended for computer science students, programmers and professionals who have already got acquainted with the basic courses and

background on discrete mathematics. It may be used as a textbook for graduate course on theoretical computer science.

The book introduces a reader to the conceptual framework for thinking about computations with the objects. The several areas of theoretical computer science are covered, including the following: type free and typed λ-calculus and combinatory logic with applications, evaluation of expressions, computations in a category. The topics, covered in the book accumulated much experience in teaching these subjects in graduate computer science courses.

A rich set of examples and exercises, including solutions, has been prepared to stimulate the self studying and to make easier the job of instructor.

♦ ♦ ♦



Wolfengagen V. E. *Combinatory logic in programming. Computations with objects through examples and exercises.* — 2-nd ed. — Moscow: «Center JurInfoR», 2003. — X+336 p. (in English)

Summary. The book is intended for computer science students, programmers and professionals who have already got acquainted with the basic courses and background on discrete mathematics. It may be used as a textbook for graduate course on theoretical computer science.

The book introduces a reader to the conceptual framework for thinking about computations with the objects. The several areas of theoretical computer science are covered, including the following: type free and typed λ-calculus and combinatory logic with applications, evaluation of expressions, computations in a category. The topics, covered in the book accumulated

much experience in teaching these subjects in graduate computer science courses.

A rich set of examples and exercises, including solutions, has been prepared to stimulate the self studying and to make easier the job of instructor.

♦ ♦ ♦

Wolfengagen V. E. *Combinatory logic in programming. Computations with objects through examples and exercises.* — 3-rd ed., revised. — Moscow: «Center JurInfoR», 2008. — X+384 p. (in Russian)

Summary. The book is intended for computer science students, programmers and professionals who have already got acquainted with the basic courses and background on discrete mathematics. It may be used as a textbook for graduate course on theoretical computer science.

The book introduces a reader to the conceptual framework for thinking about computations with the objects. The several areas of theoretical computer science are covered, including the following: type free and typed λ-calculus and combinatory logic with applications, evaluation of expressions, computations in a category. The topics, covered in the book accumulated much experience in teaching these subjects in graduate computer science courses.

A rich set of examples and exercises, including solutions, has been prepared to stimulate the self studying and to make easier the job of instructor.

♦ ♦ ♦

Wolfengagen V. E. *Methods and Means for Computations with Objects. Applicative Computational Systems.* — Moscow: JurInfoR Ltd., «Center JurlnfoR», 2004. — XVI+789 p. (in Russian)

Summary. Those models, methods and means are covered that are based on the notation of object. An approach based on the operations of application and functional abstraction is used resulting in the closed consideration of applicative computations within the elementary framework. The material covered in this book was used for delivering the various versions of courses in computer science.

The essential theoretical background corresponds to the high international level, and the basic computational ideas, notions and definitions are explicated.

The book is intended for computer science students and professionals in informatics. It may be used as a sourcebook for graduate course on theoretical computer science.

RFBR's project 03-01-14055-д.

♦ ♦ ♦

Wolfengagen V. E. *Logic. Conspectus of the lectures in formal reasoning.* — 2nd ed., completed and improved. — M.: «Center JurInfoR» Ltd., 2004. — 229 p. (in Russian)

Summary. This edition is significantly improved and completed by the elements of semantic reasoning using the classes and relations which are especially important for working out the electronic forms of information. The ways of reformulating the text of factual kind

into symbolic language allowing the application of classic logic means are considered. The techniques and ways of writing the formal reasons and their validation procedure are shown. The technique of formal logical reasonings, derivations and proofs is illustrated by a lot of examples. The ways of including the annotations (comments) into derivation are indicated which allows checking the truth or false of reasons.

This book is primary intended for the students and postgraduates of humanitarian specialties. It can be used for the initial studying of the subject and for self studying as well.

♦ ♦ ♦

Kosikov S. V. *Information Systems: Category Theory Approach.* — Moscow: JurInfoR-Press Ltd., 2006. — 96 p. (in Russian)

Summary. The book gives an account to the basic ideas related to the designing of information systems by using methods of the category theory. A detailed account is given of theoretico-categorical constructions that are used for building theoretical models and practical realization of information systems.

Considerable attention is given to how abstract machines are built as a basis for realizing information systems by means of category computational models.

The book can be used as reference material for students, post-graduate students, experts in the field of designing informational systems as well as for the application of mathematical methods in the new information technology.

♦ ♦ ♦

The applicative computational systems. Proceedings of the conference on applicative computational systems (ACS'2008), Moscow: April 29-30, 2008. / Dr. L.Yu. Ismailova (ed.). — M.: NEI Institute of Contemporary Education «JurInfoR-MGU», 2008. — 48 p. (in Russian)

Summary. Applicative computational systems, or ACSs contain the calculi of objects based on Combinatory Logic and lambda-calculus. The only thing which is essentially developing in these systems is the representation of an object. The combinatory logic contains the only metaoperator — the application, or, in other terms, the action of one object to another. The lambda-calculus contains a pair of metaoperators — the application and functional abstraction which allow binding of one variable within one object.

The objects which are generated in these systems are the fundamental entities having the following properties: (1) the number of argument places, or arity of an object is not fixed from the beginning, but evolves step by step, in interaction with other objects; (2) in developing the comprehended object one of the initial objects — the function — is applied to other object — its argument, but in other contexts they can change their roles, i.e. both the functions and arguments are the objects on equal rights; (3) a self application of the functions is allowed, i.e. an object can be applied to itself. ACSs give the grounds for the applicative approach to programming.

Applicative computing enables the combined development of a computation as a relatively self contained block using the existing blocks of computations, but all the variables in any block of computation are bound and the block itself is closed. The ACSs are used for enabling the applicative computing.

RFBR's project 08-07-06039-r.

Contents

Viacheslav **Wolfengagen**

Applicative computing
Its quarks, atoms and molecules

Technical editor *A. E. Zaitsev*

The production conforms the conditions of the
Public Health Department of the Russian Federation.
Sanitary-epidemiologic Certificate
№ 77.99.02.953.D.007462.11.05 issued by November 11, 2005

Signed to publishing 10.02.2010. Offset print. Offset paper.
Format 60x84/16. Pr. sh. 4. Copies 500. Order № .

"Center JurInfoR" Ltd.
Phone (495) 778-87-26, 971-73-96,
http://www.jurinfor.ru, e-mail: info@jurinfor.ru

Отпечатано в ОАО «Щербинская типография».
117623, г. Москва, ул. Типографская, д. 10.

Notes

Notes